Prueba objetiva de Concurrencia - Clave b (Solución)

2009-2010 - Segundo semestre

Lenguajes y Sistemas Informáticos e Ingeniería de Software

Normas

Este es un cuestionario tipo test que consta de **?? preguntas** en **?? páginas**. La puntuación total del examen es de **?? puntos**. La duración total es de **una hora y media**. El examen debe contestarse en las **hojas de respuestas**. No olvidéis rellenar **apellidos, nombre y DNI** en cada hoja de respuesta.

Sólo hay una respuesta válida por pregunta. Toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada. Toda pregunta incorrectamente contestada restará del examen una cantidad de puntos igual a la puntuación de la pregunta dividido por el número de alternativas ofrecidas en la misma.

Cuestionario

(1 punto) 1. Dado el siguiente programa implementado con la librería JCSP:

```
static private Any2OneChannel c1 = Channel.any2one();
static private Any2OneChannel c2 = Channel.any2one();
```

```
static class A implements CSProcess {
    public void run() {
        c1.out().write("A");
        String s =
            (String) c2.in().read();
        System.out.print(s);
    }
}
```

```
static class B implements CSProcess {
    public void run() {
        c2.out().write("B");
        String s =
            (String) c1.in().read();
        System.out.print(s);
    }
}
```

```
    // Programa principal
    CSProcess sistema = new Parallel(new CSProcess[] {new A(), new B()});
    sistema.run();
```

Se pide: señalar la respuesta correcta.

(a) ☐ La salida del programa es siempre `BA`.

(b) ☑ Se produce un interbloqueo y el programa no produce salida alguna.

(c) ☐ La salida del programa es `AB` o `BA`.

(d) ☐ La salida del programa es siempre `AB`.

(1 punto) 2. Dado el siguiente programa concurrente

```
static int x = 0;
```

```
static class T extends Thread {
    private int y;
    public T (int y) {
        this.y = y;
    }
```

```
    public void run() {
        int z = y;
        z = z + y; x = x + z;
    }
}
```

```
    // Programa principal
    Thread[] t = new Thread[] {new T(1), new T(2)};
    t[0].start(); t[1].start(); t[0].join(); t[1].join();
```

Se pide marcar la afirmación correcta.

(a) ☐ Es necesario asegurar exclusión mutua en el acceso a la variable `z`.

(b) ☐ Ninguna de las otras respuestas es correcta.

(c) ☑ Es necesario asegurar exclusión mutua en el acceso a la variable `x`.

(d) ☐ Es necesario asegurar exclusión mutua en el acceso al atributo `y`.

(1 punto) 3. La clase `PorTurno` implementa un protocolo de acceso a una sección crítica:

```
static int turno = 0;

static class PorTurno extends Thread {
    private int pid;

    public PorTurno(int pid) {
        this.pid = pid;
    }
```

```
    public void run() {
        while (true) {
            s();
            while (turno != pid) {}
            seccionCritica();
            turno = (turno + 1)
                    % MAX_THREADS;
        }
    }
}
```

Dado un programa concurrente con `MAX_THREADS` *threads* de la clase `PorTurno` compartiendo una variable `turno` inicializada a `0` y cada una de ellas con un índice `pid` distinto entre `0` y `MAX_THREADS-1`.

Se pide marcar la afirmación correcta.

(a) ☐ Garantizada la terminación de `s()` y `seccionCritica()`, el programa no cumple la propiedad de ausencia de inanición.

(b) ☑ Garantizada la terminación de `s()` y `seccionCritica()` el programa cumple las propiedades de exclusión mutua en `seccionCriticar()`, ausencia de interbloqueo y ausencia de inanición pero un proceso podría esperar para ejecutar `seccionCritica()` sin que los demás ejecuten `seccionCritica()` ni compitan para hacerlo.

(c) ☐ Garantizada la terminación de `s()` y `seccionCritica()`, el programa no cumple la propiedad de ausencia de interbloqueo.

(d) ☐ El programa no cumple la propiedad de exclusión mutua en `seccionCritica()`.

(1 punto) 4. Dado un programa concurrente en la que tres *threads* instancias de las clases `C`, `D` y `E` comparten una variable `n`:

```
static int n = 0;
static Semaphore s1 =
    new Semaphore(1);
static Semaphore s2 =
    new Semaphore(0);

static class C extends Thread {
    public void run() {
        s2.acquire();
        s1.acquire();
        n = 2 * n;
        s1.release();
    }
}
```

```
static class D extends Thread {
    public void run() {
        s1.acquire();
        n = n * n;
        s1.release();
    }
}

static class E extends Thread {
    public void run() {
        s1.acquire();
        n = n + 3;
        s2.release();
        s1.release();
    }
}
```

Se pide marcar el conjunto que contiene únicamente los posibles valores de la variable `n` tras la terminación de los tres threads.

(a) ☐ $\{3, 6, 18, 36\}$

(b) ☑ $\{6, 18, 36\}$

(c) ☐ No está garantizada la terminación de las tres tareas.

(d) ☐ $\{3, 6, 9, 18, 36\}$

(1 punto) 5. Dado el siguiente **CTAD** (sólo se muestran las partes necesarias):

TIPO: T_Cuatro $= (a : \mathbb{B} \times b : \mathbb{B})$

INICIAL(r): $r = (falso, falso)$

CPRE: *cierto*
S(r)
POST: $r^{ent} = (pe, se) \wedge r^{sal} = (ps, \neg se) \wedge ps = (\neg pe \wedge se) \vee (pe \wedge \neg se)$

CPRE: $r \neq (cierto, falso)$
M(r)
POST: $r^{sal} = (\neg r^{ent}.a, r^{ent}.b)$

Se pide marcar la afirmación correcta:

(a) ☐ El recurso puede pasar, a lo sumo, por 3 estados diferentes.

(b) ☑ Si el sistema consta de un recurso de este tipo, un proceso que invoca a *S* una sola vez y otro que intenta ejecutar *M* indefinidamente, el sistema no puede terminar en interbloqueo.

(c) ☐ El recurso podría estar en un estado dado y, tras ejecutarse una sola de sus acciones, continuar en el mismo estado.

(d) ☐ Si el sistema solo contiene (además de un recurso de este tipo) un único proceso que intenta ejecutar *S* indefinidamente, el sistema podría acabar en interbloqueo.

PISTA: Dibujar el grafo de estados en hoja aparte.